REPORT DOCUMENTATION PAGE

AFRL-SR-BL-TR-98-

0351

1. AGENCY USE ONLY (*Leave Blank*)	2. REPORT DATE March 1998	3. REPORT TYPE AND DATES COVERED Final Report
4. TITLE AND SUBTITLE Planning and Scheduling: Dynamic Assignment and Scheduling with Contingencies		5. FUNDING NUMBERS F49620-97-C-0013
6. AUTHOR(S) David A. Castanon, David A. Logan, Dimitri P. Bertsekas, Stephen D. Patek		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) ALPHATECH, Inc. 50 Mall Road Burlington, MA 01803		8. PERFORMING ORGANIZATION REPORT NUMBER TR-865
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Dr. Neal Glassman AFOSR/NM Directorate of Mathematics & Geosciences 110 Duncan Ave., Room B115 Bolling AFB, DC 20332-8050		10. SPONSORING / MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES		19980430 080
12a. DISTRIBUTION / AVAILABILITY STATEMENT Unlimited		12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

Planning and scheduling under uncertainty is important in many important Air Force operations. When real time monitoring of the progress of operations is available, it is possible to replan missions in response to contingencies. In this research, we investigated a class of planning and scheduling problems under uncertainty which captured important features of risky multiplatform Air Force missions.

Planning and scheduling problems under uncertainty can be solved in principle by stochastic dynamic programming techniques. These techniques require large computations for moderate problems, as they must anticipate all of the possible future events. In this research, we investigated the use of approximate stochastic dynamic programming techniques to obtain near-optimal schedules which anticipate future contingencies, and which can replan in response to contingencies. These approximations are based on techniques for obtaining estimates of the future costs associated with current decisions, using techniques such as rollout of heuristic strategies, off-line training of approximations, or approaches such as neuro-dynamic programming.

The results indicate that, in the context of the mathematical problems investigated, the performance of some approximate dynamic programming algorithms is near that of the optimal performance. Further development and evaluation will establish the viability of these techniques for Air Force mission planning problems.

14. SUBJECT TERMS			15. NUMBER OF PAGES
			16. PRICE CODE
17. SECRUTIY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UNLIMITED

NSN 7540-01-280-5500

Standard From 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18
298-102

DTIC QUALITY INSPECTED 3

EXECUTIVE SUMMARY

Planning and scheduling problems arise in many important Air Force operations. In most of these operations, information regarding the battle space for the operations is uncertain and evolves over time. Future information systems will provide the capability for monitoring the progress of operations in real-time for commanders to modify prior plans and schedules. This creates a need for fast, real-time techniques which optimize plans in the presence of uncertainty, and which anticipate contingencies which require replanning, so that the original plans can be modified with minimal impact.

In this research, we sudied a class of planning and scheduling problems motivated by the Air Force application of data collection. In this application, surveillance platforms are tasked to fly routes and collect information on targets along their routes; contingencies arise when platforms fail or the set of targets changes dynamically. We developed a mathematical abstraction of this problem as a multiplatform routing and scheduling problem on a graph with risky routes which can lead to platform destruction. In this abstraction, there is uncertainty as to whether a platform will complete its mission successfully, or be destroyed. Thus, good schedules must hedge against such contingencies, and provide the opportunity for backups, and allow for fast replanning.

Conceptually, the problem of planning and scheduling under uncertainty can be solved as a Markovian decision problem, using stochastic dynamic programming. However, this requires prohibitive amounts of computation for realistically sized problems, and thus is inadequate as a practical algorithm for real time planning or replanning. The approach pursued in this research is based on approximate stochastic dynamic programming techniques. These techniques generate estimates of the future costs of optimal strategies, and thus allow these estimates to be used in evaluating the consequences of current decisions. The resulting decision process accounts successfully for future consequences as well as present rewards, and thus results in good performance in the presence of contingencies.

The key to the research approach is obtaining accurate approximations to the future optimal costs. In this research, we focused on one powerful class of methods: rollout strategies based on simple heuristics. Rollouts are a class of solution methods inspired from the policy iteration methodology of dynamic programming and the approximate policy iteration methodology of neuro-dynamic programming. One may view a rollout algorithm as a single step of the classical policy iteration method, starting from some given easily implementable policy. such as that obtained from application of a simple heuristic. The basic idea is to use as an estimate of the future optimal cost the expected cost associated with a simple policy. The rollout

algorithm performs a policy iteration step which is guaranteed to improve on the performance of the simple policy.

Algorithms of this type have been used successfully in several dynamic programming application contexts. They have also been proposed by Tesauro [1996] in the context of simulation-based computer backgammon; the name "rollout" was introduced by Tesauro as a synonym for repeatedly playing out a given backgammon position to calculate by Monte Carlo averaging the expected game score starting from that position. Rollout algorithms were first proposed for the approximate solution of discrete optimization problems by Bertsekas and Tsitsiklis [1996], and by Bertsekas, Tsitsiklis, and Wu [1997]. The methodology used in this research is similar to the ideas in those sources, but extended to the stochastic planning situations.

One particular variation of multiplatform planning problems which was addressed in detail consisted of variations of a classical stochastic scheduling problem called the quiz problem. A simple summary of a quiz problem is as follows: Assume that there is a finite set of K locations which contain tasks of interest, of differing value. There is a single processor on which the tasks are to be scheduled. Associated with each task is a task-dependent risk that, while executing that task, the processor will be damaged and no further tasks will be processed. The objective is to find the optimal task schedule in order to maximize the expected value of the completed tasks.

In the classical quiz problem, there is no constraint on the sequences or times during which the tasks can be scheduled, and there is only a single processor. This classical problem can be viewed in terms of dynamic programming, but can more simply be viewed as a combinatorial problem, whereby we are seeking an optimal sequence in which to answer the questions. It is well-known that the optimal sequence is deterministic, and can be obtained using an interchange argument: questions should be answered in decreasing order of an index depending on task values and the probabilities of task completion. However, minor changes in the structure of the problem lead to much more complicated optimal strategies. These changes include time limits on the total number of tasks which can be processed, time windows for scheduling each task, precedence constraints or multiple processors.

Our investigations focused on developing approximate dynamic programming algorithms for these extensions of classical quiz problems. The work presented in Appendix A considers a variety of extensions deterministic and stochastic versions of quiz problems involving a single processor, and compares the peformance of rollout algorithms with that of common heuristics and with the optimal stochastic dynamic programming solution. The results demonstrate that the rollout algorithms perform significantly better than the underlying heuristics and, under some circumstances within 5-10% of the optimal solution.

The next step in our research was to investigate extensions to multiplatform scheduling problems. A mathematical statement of this problem can be developed for platforms constrained to moving on graphs, with tasks located at nodes in the graph. This framework incorporates precedence constraints as well as risks. Our investigations of these problems are documented in the working paper included in Appendix B. Our initial approaches were based on developing approximations to the optimal cost to go using constraint modifications and certainty equivalent deterministic approximations. We have yet to implement the equivalent of rollout strategies for this class of problems. The results in Appendix B show that the certainty equivalent approximations proposed have very poor performance when used as approximations to the stochastic cost-to-go, a fact which surprised us and suggests that alternative approaches are required.

The research described in this executive summary and the two appendices was performed by Prof. David Castañon, Mr. David Logan, Prof. Dimitri Bertsekas, and Dr. Stephen Patek. The paper in Appendix A has been submitted for publication to the 1998 Conference on Decision and Control, and to the journal *Heuristics*.

APPENDIX A

ROLLOUT ALGORITHMS FOR STOCHASTIC SCHEDULING PROBLEMS[1]

by

Dimitri P. Bertsekas[2] and David A. Castañon[3]

Abstract

Stochastic scheduling problems are difficult stochastic control problems with combinatorial decision spaces. In this paper we focus on a class of stochastic scheduling problems, the quiz problem and its variations. We discuss the use of heuristics for their solution, and we propose rollout algorithms based on these heuristics which approximate the stochastic dynamic programming algorithm. We show how the rollout algorithms can be implemented efficiently, and we delineate circumstances under which they are guaranteed to perform better than the heuristics on which they are based. We also show computational results which suggest that the performance of the rollout policies is near-optimal, and is substantially better than the performance of their underlying heuristics.

[1] This work was supported in part by the Air Force Office of Scientific Research under grant no. F49620-97-C-0013

[2] Department of Electrical Engineering and Computer Science, M. I. T., Cambridge, Mass., 02139.

[3] Department of Electrical Engineering, Boston University, and ALPHATECH, Inc., Burlington, Mass., 01803.

1. INTRODUCTION

Consider the following variation of a planning problem: There is a finite set of K locations which contain tasks of interest, of differing value. There is a single processor on which the tasks are to be scheduled. Associated with each task is a task-dependent risk that, while executing that task, the processor will be damaged and no further tasks will be processed. The objective is to find the optimal task schedule in order to maximize the expected value of the completed tasks.

The above is an example of a class of stochastic scheduling problems known in the literature as *quiz problems* (see Bertsekas [1995], Ross [1983], or Whittle [1982]). The simplest form of this problem involves a quiz contest where a person is given a list of N questions and can answer these questions in any order he or she chooses. Question i will be answered correctly with probability p_i, and the person will then receive a reward v_i. At the first incorrect answer, the quiz terminates and the person is allowed to keep his or her previous rewards. The problem is to choose the ordering of questions so as to maximize expected rewards.

The problem can be viewed in terms of dynamic programming (DP for short), but can more simply be viewed as a deterministic combinatorial problem, whereby we are seeking an optimal sequence in which to answer the questions. It is well-known that the optimal sequence is deterministic, and can be obtained using an interchange argument; questions should be answered in decreasing order of $p_i v_i/(1-p_i)$. This will be referred to as the *index policy*. An answer order is optimal if and only if it corresponds to an index policy. Another interesting simple policy for the quiz problem is the *greedy policy*, which answers questions in decreasing order of their expected reward $p_i v_i$. A greedy policy is suboptimal, essentially because it does not consider the future opportunity loss resulting from an incorrect answer.

Unfortunately, with only minor changes in the structure of the problem, the optimal solution becomes much more complicated (although DP and interchange arguments are still relevant). Examples of interesting and difficult variations of the problem involve one or more of the following characteristics:

(a) A limit on the maximum number of questions that can be answered, which is smaller than the number of questions N. To see that the index policy is not optimal anymore, consider the case where there are two questions, only one of which may be answered. Then it is optimal to use the greedy policy rather than the index policy.

(b) A time window for each question, which constrains the set of time slots when each question may be answered. Time windows may also be combined with the option to refuse answering a question at a given period, when either no question is available during the period, or answering any one of the available questions involves excessive risk.

(c) Precedence constraints, whereby the set of questions that can be answered in a given time slot depends on the immediately preceding question, and possibly on some earlier answered questions.

(d) Sequence-dependent rewards, whereby the reward from answering correctly a given question depends on the immediately preceding question, and possibly on some questions answered earlier.

It is clear that the quiz problem variants listed above encompass a very large collection of practical scheduling problems. The version of the problem with time windows and precedence constraints relates to vehicle routing problems (involving a single vehicle). The version of the problem with sequence-dependent rewards, and a number of questions that is equal to the maximum number of answers relates to the traveling salesman problem. Thus, in general, it is very difficult to solve the variants described above exactly.

An important feature of the quiz problem, which is absent in the classical versions of vehicle routing and traveling salesman problems is that *there is a random mechanism for termination of the quiz*. Despite the randomness in the problem, however, in all of the preceding variants, there is an *optimal open-loop policy*, i.e., an optimal order for the questions that does not depend on the random outcome of the earlier questions. The reason is that we do not need to plan the answer sequence following the event of an incorrect answer, because the quiz terminates when this event occurs. Thus, we refer to the above variations of the quiz problem as *deterministic quiz problems.*

There are variants of the quiz problem where the optimal order to answer questions depends on random events. Examples of these are:

(e) There is a random mechanism by which the quiz taker may miss a turn, i.e., be denied the opportunity to answer a question at a given period, but may continue answering questions at future time periods.

(f) New questions can appear and/or old questions can disappear in the course of the quiz according to some random mechanism. A similar case arises when the start and end of the time windows can change randomly during the quiz.

(g) There may be multiple quiz takers that answer questions individually, and drop out of the quiz upon their own first error, while the remaining quiz takers continue to answer questions.

(h) The quiz taker may be allowed multiple chances, i.e., may continue answering questions up to a given number of errors.

(i) The reward for answering a given question may be random and may be revealed to the quiz taker at various points during the course of the quiz.

The variants (e)-(i) of the quiz problem described above require a genuinely stochastic formulation as Markovian decision problems. We refer to these variations in the paper as *stochastic quiz problems.* They can be solved exactly only with DP, but their optimal solution is prohibitively difficult. This is because the states over which DP must be executed are subsets of questions, and the number of these subsets increases exponentially with the number of questions.

In this paper, we develop suboptimal solution approaches for deterministic and stochastic quiz problems that are computationally tractable. In particular, we focus on rollout algorithms, a class of suboptimal solution methods inspired from the policy iteration methodology of DP and the approximate policy iteration methodology of neuro-dynamic programming (NDP for short). One may view a rollout algorithm as a single step of the classical policy iteration method, starting from some given easily implementable policy. Algorithms of this type have been sporadically proposed in several DP application contexts. They have also been proposed by Tesauro [1996] in the context of simulation-based computer backgammon (the name "rollout" was introduced by Tesauro as a synonym for repeatedly playing out a given backgammon position to calculate by Monte Carlo averaging the expected game score starting from that position).

Rollout algorithms were first proposed for the approximate solution of discrete optimization problems by Bertsekas and Tsitsiklis [1996], and by Bertsekas, Tsitsiklis, and Wu [1997], and the methodology developed here for the quiz problem strongly relates to the ideas in these sources. Generally, rollout algorithms are capable of magnifying the effectiveness of any given heuristic algorithm through sequential application. This is due to the policy improvement mechanism of the underlying policy iteration process.

In the next section, we introduce rollout algorithms for deterministic quiz problems, where the optimal order for the questions from a given period onward does not depend on earlier random events. In Section 3, we provide computational results indicating that rollout algorithms can improve impressively on the performance of their underlying heuristics. In Sections 4 and 5, we extend the rollout methodology to stochastic quiz problems [cf. variants (e)-(i) above], that require the use of stochastic DP for their optimal solution. Here we introduce the idea of *multiple scenaria* for the future uncertainty starting from a given state, and we show how these scenaria can be used to construct an approximation to the optimal value function of the problem using NDP techniques and a process of *scenario aggregation*. Finally, in Section 6, we provide computational results using rollout algorithms for stochastic quiz problems.

2. ROLLOUT ALGORITHMS FOR DETERMINISTIC QUIZ PROBLEMS

Consider a variation of a quiz problem of the type described in (a)-(c) above. Let N denote the number of questions available, and let M denote the maximum number of questions which may be attempted. Associated with each question i is a value v_i, and a probability of successfully answering that question p_i. Assume that there are constraints such as time windows or precedence constraints which restrict the possible question orders. Denote by $V(i_1,\ldots,i_M)$ the expected reward of a feasible question order $(i_1,\ldots,i_M)$:

$$V(i_1,\ldots,i_M)=p_{i_1}\big(v_{i_1}+p_{i_2}\big(v_{i_2}+p_{i_3}(\cdots+p_{i_M}v_{i_M})\cdots\big)\big). \tag{2.1}$$

For an infeasible question order $(i_1,\ldots,i_M)$, we use the convention

$$V(i_1,\ldots,i_M)=-\infty.$$

The classical quiz problem is the case where $M=N$, and all question orders are feasible. In this case, the optimal solution is simply obtained by using an interchange argument. Let i and j be the kth and $(k+1)$st questions in an optimally ordered list

$$L=(i_1,\ldots,i_{k-1},i,j,i_{k+2},\ldots,i_N).$$

Consider the list

$$L'=(i_1,\ldots,i_{k-1},j,i,i_{k+2},\ldots,i_N)$$

obtained from L by interchanging the order of questions i and j. We compare the expected rewards of L and L'. We have

$$\begin{aligned}E\{\text{reward of } L\}&=E\big\{\text{reward of } \{i_1,\ldots,i_{k-1}\}\big\}\\&\quad+p_{i_1}\cdots p_{i_{k-1}}(p_iv_i+p_ip_jv_j)\\&\quad+p_{i_1}\cdots p_{i_{k-1}}p_ip_jE\big\{\text{reward of } \{i_{k+2},\ldots,i_N\}\big\}\\E\{\text{reward of } L'\}&=E\big\{\text{reward of } \{i_1,\ldots,i_{k-1}\}\big\}\\&\quad+p_{i_1}\cdots p_{i_{k-1}}(p_jv_j+p_jp_iv_i)\\&\quad+p_{i_1}\cdots p_{i_{k-1}}p_jp_iE\big\{\text{reward of } \{i_{k+2},\ldots,i_N\}\big\}.\end{aligned}$$

Since L is optimally ordered, we have

$$E\{\text{reward of } L\} \geq E\{\text{reward of } L'\},$$

so it follows from these equations that

$$p_i v_i + p_i p_j v_j \geq p_j v_j + p_j p_i v_i$$

or equivalently

$$\frac{p_i v_i}{1 - p_i} \geq \frac{p_j v_j}{1 - p_j}.$$

It follows that to maximize expected rewards, questions should be answered in decreasing order of $p_i v_i/(1 - p_i)$, which yields the index policy.

Unfortunately, the above argument breaks down when either $M < N$, or there are constraints on the admissibility of sequences due to time windows or precedence constraints. For these cases, we can still use heuristics such as the index policy or the greedy policy, but they will not provide optimal performance.

Consider a heuristic algorithm, which given a *partial schedule* $P = (i_1, \dots, i_k)$ of distinct questions constructs a *complementary schedule* $\overline{P} = (i_{k+1}, \dots, i_M)$ of distinct questions such that $P \cap \overline{P} = \emptyset$. The heuristic algorithm is referred to as the *base heuristic*. We define the *heuristic reward* of the partial schedule P as

$$H(P) = V(i_1, \dots, i_k, i_{k+1} \dots, i_M). \tag{2.2}$$

If $P = (i_1, \dots, i_M)$ is a complete solution, by convention the heuristic reward of P is the true expected reward $V(i_1, \dots, i_M)$.

Given a base heuristic, the corresponding *rollout algorithm* constructs a complete schedule in M iterations, one question per iteration. The rollout algorithm can be described as follows:

At the 1st iteration it selects question i_1 according to

$$i_1 = \arg \max_{i=1,\dots,N} H(i), \tag{2.3}$$

and at the kth iteration $(k > 1)$ it selects i_k according to

$$i_k = \arg \max_{\{i \mid i \neq i_1, \dots, i_{k-1}\}} H(i_1, \dots, i_{k-1}, i), \qquad k = 2, \dots, M. \tag{2.4}$$

Thus a rollout policy involves $N + (N - 1) + \cdots + (N - M) = O(MN)$ applications of the base heuristic and corresponding calculations of expected reward of the form (2.1). While this is a significant increase over the calculations required to apply the base heuristic and compute its expected reward, the rollout policy is still computationally tractable. In particular, if the running time of the base heuristic is polynomial, so is the running time of the corresponding rollout algorithm. On the other hand, it will be shown shortly that the expected reward of the rollout policy is at least as large as the one of the base heuristic.

As an example of a rollout algorithm, consider the special variant (a) of the quiz problem in the preceding section, where at most M out of N questions may be answered and there are no time windows or other complications. Let us use as base heuristic the index heuristic, which given a partial schedule $(i_1, \ldots, i_k)$, attempts the remaining questions according to the index policy, in decreasing order of $p_i v_i/(1-p_i)$. The calculation of $H(i_1, \ldots, i_k)$ is done using Eq. (2.1), once the questions have been sorted in decreasing order of index. The corresponding rollout algorithm, given $(i_1, \ldots, i_{k-1})$ selects i, calculates $H(i_1, \ldots, i_{k-1}, i)$ for all $i \neq i_1, \ldots, i_{k-1}$, using Eq. (2.1), and then optimizes this expression over i to select i_k.

Note that one may use a different heuristic, such as the greedy heuristic, in place of the index heuristic. There are also other possibilities for base heuristics. For example, one may first construct a complementary schedule using the index heuristic, and then try to improve this schedule by using a 2-OPT local search heuristic, that involves exchanges of positions of pairs of questions. One may also use multiple heuristics, which produce heuristic values $H_j(i_1, \ldots, i_k)$, $j = 1, \ldots, J$, of a generic partial schedule $(i_1, \ldots, i_k)$, and then combine them into a "superheuristic" that gives the maximal value

$$H(i_1, \ldots, i_k) = \max_{j=1,\ldots,J} H_j(i_1, \ldots, i_k).$$

An important question is whether the rollout algorithm performs at least as well as its base heuristic when started from the initial partial schedule. This can be guaranteed if the base heuristic is *sequentially consistent*. By this we mean that the heuristic has the following property:

Suppose that starting from a partial schedule

$$P = (i_1, \ldots, i_{k-1}),$$

the heuristic produces the complementary schedule

$$\overline{P} = (i_k, \ldots, i_M).$$

Then starting from the partial schedule

$$P^+ = (i_1, \ldots, i_{k-1}, i_k),$$

the heuristic produces the complementary schedule

$$\overline{P}^{+} = (i_{k+1}, \ldots, i_M).$$

As an example, it can be seen that the index and the greedy heuristics, discussed earlier, are sequentially consistent. This is a manifestation of a more general property: many common base heuristics of the greedy type are by nature sequentially consistent. It may be verified, based on Eq. (2.4), that a sequentially consistent rollout algorithm keeps generating the same schedule $P \cup \overline{P}$, up to the point where by examining the alternatives in Eq. (2.4) and by calculating their heuristic rewards, it discovers a better schedule. As a result, sequential consistency guarantees that the reward of the schedules $P \cup \overline{P}$ produced by the rollout algorithm is monotonically nonincreasing; that is, we have

$$H(P^{+}) \leq H(P)$$

at every iteration. For further elaboration of the sequential consistency property, we refer to the paper by Bertsekas, Tsitsiklis, and Wu [1997], which also discusses some underlying connections with the policy iteration method of dynamic programming.

A condition that is more general than sequential consistency is that the algorithm be *sequentially improving*, in the sense that at each iteration there holds

$$H(P^{+}) \leq H(P).$$

This property also guarantees that the rewards of the schedules produced by the rollout algorithm are monotonically nonincreasing. The paper by Bertsekas, Tsitsiklis, and Wu [1997] discusses situations where this property holds, and shows that with fairly simple modification, a rollout algorithm can be made sequentially improving.

There are a number of variations of the basic rollout algorithm described above. In particular, we may incorporate *multistep lookahead* or *selective depth lookahead* into the rollout framework. An example of a rollout algorithm with m-step lookahead operates as follows: at the kth iteration we augment the current partial schedule $P = (i_1, \ldots, i_{k-1})$ with all possible sequences of m questions $i \neq i_1, \ldots, i_{k-1}$. We run the base heuristic from each of the corresponding augmented partial schedules, we select the m-question sequence with maximum heuristic reward, and then augment the current partial schedule P with the first question in this sequence. An example of a rollout algorithm with *selective* two-step lookahead operates as follows: at the kth iteration we start with the current partial schedule $P = (i_1, \ldots, i_{k-1})$, and we run the base heuristic starting from each partial schedule $(i_1, \ldots, i_{k-1}, i)$ with $i \neq i_1, \ldots, i_{k-1}$. We then form the subset $\overline{I}$ consisting of the n questions $i \neq i_1, \ldots, i_k$ that correspond to the n best complete schedules thus obtained. We run the base heuristic starting from each of the partial schedules $(i_1, \ldots, i_{k-1}, i, j)$ with $i \in \overline{I}$ and $j \neq i_1, \ldots, i_{k-1}, i$, and obtain a corresponding complete schedule. We then select as next question i_k of the rollout schedule the question $i \in \overline{I}$ that corresponds to a maximal reward schedule. Note that by choosing the number n to be smaller than the maximum possible, $N - k + 1$, we can reduce substantially the computational requirements of the two-step lookahead.

3. COMPUTATIONAL EXPERIMENTS WITH DETERMINISTIC QUIZ PROBLEMS

In order to explore the performance of rollout algorithms for deterministic scheduling, we conducted a series of computational experiments involving the following seven algorithms:

(1) The optimal stochastic dynamic programming algorithm.

(2) The greedy heuristic, where questions are ranked in order of decreasing $p_i v_i$, and, for each stage k, the feasible unanswered question with the highest ranking is selected.

(3) The index heuristic, where questions are ranked in order of decreasing $p_i v_i/(1 - p_i v_i)$, and for each stage k, the feasible unanswered question with the highest ranking is selected.

(4) The one-step rollout policy based on the greedy heuristic, where, at each stage k, for every feasible unanswered question i_k and prior sequence $i_1, \ldots, i_{k-1}$, the question is chosen according to the rollout rule (2.4), wher the function H uses the greedy heuristic as the base policy.

(5) The one-step rollout policy based on the index heuristic, where the function H in (2.4) uses the index heuristic as the base policy,

(6) The selective two-step lookahead rollout policy based on the greedy heuristic. At the k-th stage, the base heuristic is used in a one-step rollout to select the best four choices for the current question among the admissible choices. For each of these choices at stage k, the feasible continuations at stage $k + 1$ are evaluated using the greedy heuristic to complete the schedule. The choice at stage k is then selected from the sequence with the highest evaluation.

(7) The selective two-step lookahead rollout policy based on the index heuristic.

The problems selected for evaluation involve 20 possible questions and 20 stages, which are small enough so that exact solution using dynamic programming is possible. Associated with each question is a sequence of times, determined randomly for each experiment, when that question can be attempted. Floating point values were assigned randomly to each question from 1 to 10 in each problem instance. The probabilities of successfully answering each question were also chosen randomly, between a specified lower bound and 1.0. In order to evaluate the performance of the last six algorithms, each suboptimal algorithm was simulated 10,000 times, using independent event sequences determining which questions were answered correctly.

Our experiments focused on the effects of two factors on the relative performance of the different algorithms:

(a) The lower bound on the probability of successfully answering a question, which varied from 0.2 to 0.8

(b) The average percent of questions which can be answered at any one stage, which ranged from 10% to 50%.

The first set of experiments fixed the average percentage of questions which can be answered at a single stage to 10%, and varied the lower bound on the probability of successfully answering a question across four conditions: 0.2, 0.4, 0.6 and 0.8. For each experimental condition, we generated 30 independent problems and solved them, and evaluated the corresponding performance using 10,000 Monte Carlo runs. We computed the average performance across the 30 problems, and compared this performance with the performance obtained using the stochastic dynamic programming algorithm.

Table 1 shows the results of our experiments. The average performance of the greedy and index heuristics in each condition are expressed in terms of the percentage of the optimal performance. For low probability of success, both heuristics obtain less than half of the performance of the optimal algorithm. The table also illustrates the improvement in performance obtained by both the one-step rollout and the selective two-step rollout algorithms, expressed in terms of percentage of the optimal performance. Thus, for lower bound 0.2, the average performance across 30 problems achieved by the greedy heuristic was 41% of optimal, whereas the average performance of the one-step rollout with the greedy heuristic as a base policy achieved on average 75% of the optimal performance, which was a 34% improvement. Furthermore, the two-step selective rollout achieved on average 81% of the optimal performance.

The results in Table 1 show that one-step rollouts significantly improve the performance of both the greedy and the index heuristics in these difficult stochastic combinatorial problems. The results also illustrate that the performance of the simple heuristics improves as the average

Minimum Probability of Success	0.2	0.4	0.6	0.8
Greedy Heuristic	41%	50%	61%	76%
Improvement by One-step Rollout	34%	32%	27%	14%
Improvement by Two-step Rollout	40%	34%	27%	14%
Index Heuristic	43%	53%	66%	80%
Improvement by One-step Rollout	34%	30%	23%	10%
Improvement by Two-step Rollout	38%	33%	24%	11%

Table 1: Performance of the different algorithms as the minimum probability of success of answering a question varies. The numbers reported are percentage of the performance of the optimal dynamic programming solution achieved, averaged across 30 independent problems.

probability of success increases, thereby reducing the potential advantage of rollout strategies. Even in these unfavorable cases, the rollout strategies improved performance levels by at least 10% of the optimal policy.

For the size of problems tested in these experiments, the advantages of using a two-step selective lookahead rollout were small. In many cases, the performances of the one-step rollout and the two-step selective lookahead rollout were identical. Nevertheless, for selected difficult individual problems, the two-step selective lookahead rollout improved performance by as much as 40% of the optimal strategy over the level achieved by the one-step rollout with the same base heuristic.

The second set of experiments fixed the lower bound on the probability of successfully answering a question to 0.2, and varied the average percent of questions which can be answered at any one stage across 3 levels: 10%, 30% and 50%. As before, we generated 30 independent problems and evaluated the performance of each algorithm on each problem instance. The results of these experiments are summarized in Table 2. As before, the performance of the greedy and index heuristics improves as the experimental condition approaches the standard conditions of the quiz problem, where 100% of the questions can be answered at any time. The results confirm the trend first seen in Table 1: even in cases where the heuristics achieve good performance,

Problem Density	0.1	0.3	0.5
Greedy Heuristic	41%	58%	76%
Improvement by One-step Rollout	34%	28%	15%
Improvement by Two-step Rollout	40%	32%	16%
Index Heuristic	43%	68%	85%
Improvement by One-step Rollout	34%	22%	8%
Improvement by Two-step Rollout	38%	24%	9%

Table 2: Performance of the different algorithms as the average number of questions per period increases. The numbers reported are percentage of the performance of the optimal dynamic programming solution achieved, averaged across 30 independent problems.

rollout strategies offer significant performance gains.

The results in Tables 1 and 2 suggest that the advantages of rollout strategies over the greedy and index heuristics increase proportionately with the risk involved in the problem. By constructing a feasible strategy for the entire horizon for evaluating the current decision, rollout strategies account for the limited future accessibility of questions, and compute tradeoffs between future accessibility and the risk of the current choice. In contrast, myopic strategies such as the greedy and index heuristics do not account for future access to questions, and thus are forced to make risky choices when no other alternatives are present. Thus, as the risk of missing a question increases and the average accessibility of questions decreases, rollout strategies achieve nearly double the performance of the corresponding myopic heuristics.

4. ROLLOUT ALGORITHMS FOR STOCHASTIC QUIZ PROBLEMS

We now consider variants of the quiz problem where there is no optimal policy that is open-loop. The situations (e)-(i) given in Section 1 provide examples of quiz problems of this type. We can view such problems as stochastic DP problems. Their exact solution, however, is prohibitively expensive.

Let us state a quiz problem in the basic form of a dynamic programming problem, where we have the stationary discrete-time dynamic system

$$x_{k+1} = f_k(x_k, u_k, w_k), \qquad k = 0, 1, \ldots, N, \tag{4.1}$$

that evolves over T time periods. Here x_k is the state taking values in some set, u_k is the control to be selected from a finite set $U_k(x_k)$, w_k is a random disturbance, and f_k is a given function. We assume that the disturbance w_k, $k = 0, 1, \ldots$, has a given probability distribution that depends explicitly only on the current state and control. The one-stage cost function is denoted by $g_k(x, u, w)$.

To apply the rollout framework, we need to have a base policy for making a decision at each state-time pair (x_k, k). We view this policy as a sequence of feedback functions $\pi = \{\mu_0, \mu_1, \ldots, \mu_T\}$, which at time k maps a state x_k to a control $\mu_k(x_k) \in U_k(x_k)$. The cost-to-go of π starting from a state-time pair (x_k, k) will be denoted by

$$J_k(x_k) = E\left\{\sum_{i=k}^{T-1} g_i\big(x_i, \mu_i(x_i), w_i\big)\right\}. \tag{4.2}$$

The cost-to-go functions J_k satisfy the following recursion of dynamic programming (DP for short)

$$J_k(x) = E\big\{g\big(x, \mu_k(x), w\big) + J_{k+1}\big(f\big(x, \mu_k(x), w\big)\big)\big\}, \qquad k = 0, 1, \ldots \tag{4.3}$$

with the initial condition

$$J_T(x) = 0.$$

The *rollout policy based on* π is denoted by $\overline{\pi} = \{\overline{\mu}_0, \overline{\mu}_1, \ldots\}$, and is defined through the operation

$$\overline{\mu}_k(x) = \arg\min_{u \in U(x)} E\big\{g(x, u, w) + J_{k+1}\big(f(x, u, w)\big)\big\}, \qquad \forall\, x,\ k = 0, 1, \ldots. \tag{4.4}$$

Thus the rollout policy is a one step-lookahead policy, with the optimal cost-to-go approximated by the cost-to-go of the base policy. This amounts essentially to a single step of the method of policy iteration. Indeed using standard policy iteration arguments, one can show that the rollout policy $\overline{\pi}$ is an improved policy over the base policy π.

In practice, one typically has a method or algorithm to compute the control $\mu_k(x)$ of the base policy, given the state x, but the corresponding cost-to-go functions J_k may not be known in closed form. Then the exact or approximate computation of the rollout control $\overline{\mu}_k(x)$ using Eq. (4.4) becomes an important and nontrivial issue, since we need for all $u \in U(x)$ the value of

$$Q_k(x, u) = E\big\{g(x, u, w) + J_{k+1}\big(f(x, u, w)\big)\big\}, \tag{4.5}$$

known as the *Q-factor* at time k. Alternatively, for the computation of $\overline{\mu}_k(x)$ we need the value of the cost-to-go

$$J_{k+1}\big(f(x,u,w)\big)$$

at all possible next states $f(x,u,w)$.

In favorable cases, it is possible to compute the cost-to-go $J_k(x)$ of the base policy π for any time k and state x. An example is the variant of the quiz problem discussed in Sections 2 and 3, where the base policy is an open-loop policy that consists of the schedule generated by the index policy or the greedy policy. The corresponding cost-to-go can then be computed using Eq. (2.1). In general, however, the computation of the cost-to-go of the base policy may be much more difficult. In particular, when the number of states is very large, the DP recursion (4.3) may be infeasible.

A conceptually straightforward approach for computing the rollout control at a given state x and time k is to use Monte Carlo simulation. This was suggested by Tesauro [TeG96] in the context of backgammon. To implement this approach, we consider all possible controls $u \in U(x)$ and we generate a "large" number of simulated trajectories of the system starting from x, using u as the first control, and using the policy π thereafter. Thus a simulated trajectory has the form

$$x_{i+1} = f\big(x_i, \mu_i(x_i), w_i\big), \qquad i = k+1, \ldots, T-1,$$

where the first generated state is

$$x_{k+1} = f(x, u, w_k),$$

and each of the disturbances $w_k, \ldots, w_{T-1}$ is an independent random sample from the given distribution. The costs corresponding to these trajectories are averaged to compute an approximation $\tilde{Q}_k(x,u)$ to the Q-factor $Q_k(x,u)$. The approximation becomes increasingly accurate as the number of simulated trajectories increases. Once the approximate Q-factor $\tilde{Q}_k(x,u)$ corresponding to each control $u \in U(x)$ is computed, we can obtain the (approximate) rollout control $\tilde{\mu}_k(x)$ by the minimization

$$\tilde{\mu}_k(x) = \arg\min_{u \in U(x)} \tilde{Q}_k(x,u).$$

Unfortunately, this method suffers from the excessive computational overhead of the Monte Carlo simulation. We are thus motivated to consider approximations that involve reduced overhead, and yet capture the essense of the basic rollout idea. We describe next an approximation approach of this type, and in the following section, we discuss its application to stochastic scheduling problems.

Approximation Using Scenaria

Let us suppose that we approximate the cost-to-go of the base policy π using *certainty equivalence*. In particular, given a state x_k at time k, we fix the remaining disturbances at some nominal values $\overline{w}_k, \overline{w}_{k+1}, \ldots, \overline{w}_{T-1}$, and we generate a state and control trajectory of the system using the base policy π starting from x_k and time k. The corresponding cost is denoted by $\tilde{J}_k(x_k)$, and is used as an approximation to the true cost $J_k(x_k)$. The approximate rollout control based on π is given by

$$\tilde{\mu}_k(x) = \arg \min_{u \in U(x)} E\{g(x_k, u, w) + \tilde{J}_{k+1}\big(f(x_k, u, w)\big)\}.$$

We thus need to run π from all possible next states $f(x_k, u, w)$ and evaluate the corresponding approximate cost-to-go $\tilde{J}_{k+1}\big(f(x_k, u, w)\big)$ using a single state-control trajectory calculation based on the nominal values of the uncertainty.

The certainty equivalent approximation involves a single nominal trajectory of the remaining uncertainty. To strengthen this approach, it is natural to consider multiple trajectories of the uncertainty, called *scenaria*, and to construct an approximation to the relevant Q-factors that involves, for every one of the scenaria, the cost of the base policy π. Mathematically, we assume that we have a method, which at each state x_k, generates M uncertainty sequences

$$w^m(x_k) = (w_k^m, w_{k+1}^m, \ldots, w_{T-1}^m), \qquad m = 1, \ldots, M.$$

The sequences $w^m(x_k)$ are the scenaria at state x_k. The cost $J_k(x_k)$ of the base policy is approximated by

$$\tilde{J}_k(x_k, r) = r_0 + \sum_{m=1}^{M} r_m C_m(x_k), \tag{4.6}$$

where $r = (r_0, r_1, \ldots, r_M)$ is a vector of parameters to be determined, and $C_m(x_k)$ is the cost corresponding to an occurence of the scenario $w^m(x_k)$, when starting at state x_k and using the base policy. We may interpret the parameter r_m as an "aggregate weight" that encodes the aggregate effect on the cost-to-go function of the base policy of uncertainty sequences that are similar to the scenario $w^m(x_k)$. We will assume for simplicity that r does not depend on the time index k or the state x_k. However, there are interesting possibilities for allowing a dependence of r on k or x_k (or both), with straightforward changes in the following methodology. Note that, if $r_0 = 0$, the approximation (4.6) may be also be viewed as *limited simulation approach*, based on just the M scenaria $w^m(x_k)$, and using the weights r_m as "aggregate probabilities."

Given the parameter vector r, and the corresponding approximation $\tilde{J}_k(x_k, r)$ to the cost of the base policy, as defined above, a corresponding approximate rollout policy is determined by

$$\tilde{\mu}_k(x) = \arg \min_{u \in U(x)} \tilde{Q}_k(x, u, r), \tag{4.7}$$

where

$$\tilde{Q}_k(x,u,r) = E\{g(x,u,w) + \tilde{J}_{k+1}\big(f(x,u,w),r\big)\} \tag{4.8}$$

is the approximate Q-factor. We envision here that the parameter r will be determined by an off-line "training "process and it will then be used for calculating on-line the approximate rollout policy as above.

One may use standard methods of Neuro-Dynamic Programming (NDP for short) to train the parameter vector r. In particular, we may view the approximating function $\tilde{J}_k(x_k,r)$ of Eq. (4.6) as a linear feature-based architecture where the scenaria costs $C_m(x_k)$ are the features at state x_k. One possibility is to use a straightforward least squares fit of $\tilde{J}_k(x_k,r)$ to random sample values of the cost-to-go $J_k(x_k)$. These sample values may be obtained by Monte-Carlo simulation, starting from a representative subset of states. Another possibility is to use Sutton's TD(λ). We refer to the books by Bertsekas and Tsitsiklis [BeT96] and Barto and Sutton [BaS97], and the survey by Barto et. al. [BBS95] for extensive accounts of training methods and relating techniques.

We finally mention a variation of the scenario-based approximation method, whereby only a portion of the future uncertain quantities are fixed at nominal scenario values, while the remaining uncertain quantities are explicitly viewed as random. The cost of scenario m at state x_k is now a random variable, and the quantity $C_m(x_k)$ used in Eq. (4.6) should be the *expected* cost of this random variable. This variation is appropriate and makes practical sense as long as the computation of the corresponding expected scenaria costs $C_m(x_k)$ is convenient.

5. ROLLOUT ALGORITHMS FOR STOCHASTIC QUIZ PROBLEMS

We now apply the rollout approach based on certainty equivalence and scenaria to variants of the quiz problem where there is no optimal policy that is open-loop, such as the situations (e)-(i) given in Section 1. The state after questions $i_1,\ldots,i_k$ have been successfully answered, is the current partial schedule $(i_1,\ldots,i_k)$, and possibly the list of surviving quiz takers [in the case where there are multiple quiz takers, as in variant (g) of Section 1]. A scenario at this state corresponds to a (deterministic) sequence of realizations of some of the future random quantities, such as:

(1) The list of turns that will be missed in answer attempts from time k onward; this is for the case of variant (e) in Section 1.

(2) The list of new questions that will appear and old questions that will disappear from time k onward; this is for the case of variant (f) in Section 1.

(3) The specific future times at which the surviving quiz takers will drop out of the quiz; this is for the case of variant (g) in Section 1.

Given any scenario of this type at a given state, and a base heuristic such as an index or a greedy policy, the corresponding value of the heuristic [cf. the cost $C_m(x_k)$ in Eq. (4.6)] can be easily calculated. The approximate value of the heuristic at the given state can be computed by weighing the values of all the scenaria using a weight vector r, as in Eq. (4.6). In the case of a single scenario, a form of certainty equivalence is used, whereby the value of the scenario at a given state is used as the (approximate) value of the heuristic starting from that state. In the next section we present computational results for the case of a problem, which is identical to the one tested in Section 3, but a turn may be missed with a certain probability.

6. COMPUTATIONAL EXPERIMENTS WITH STOCHASTIC QUIZ PROBLEMS

The class of quiz problems which we used in our computational experiments are similar to the problems used in Section 3, with the additional feature that an attempt to answer a question can be blocked with a prespecified probability, corresponding to the case of variant (e) in Section 1. The problems involve 20 questions and 20 time periods, where each question has a prescribed set of times where it can be attempted. The result of a blocking event is a loss of opportunity to answer any question at that stage. Unanswered questions can be attempted in future stages, until a wrong answer is obtained.

In order to evaluate the performance of the base policy for rollout algorithms, we use a particular version of the scenaria approach described previously. Assume that the blocking probability is denoted by P_b. At any stage k, given M stages remaining and this blocking probability, the equivalent scenario duration is computed as the smallest integer greater than or equal to the expected number of stages remaining:

$$T_e = \lceil P_b * (M - k) \rceil$$

Using this horizon, the expected cost of a base heuristic is computed as the cost incurred for an equivalent deterministic quiz problem starting with the current state, with remaining duration T_e. The cost of the strategy obtained by the base policy is approximated using the resulting value function for this horizon, as computed by Eq. (2.1).

As in Section 4, we used seven algorithms in our experiments:

1. The optimal stochastic dynamic programming algorithm.

2. The greedy heuristic, where questions are ranked in decreasing $p_i v_i$, and, for each stage k, the feasible unanswered question with the highest ranking is selected.

3. The index heuristic, where questions are ranked by decreasing $p_i v_i/(1 - p_i v_i)$, and for each stage k, the feasible unanswered question with the highest ranking is selected.

4. The one-step rollout policy based on the greedy heuristic and certainty equivalence policy evaluation, where, at each stage k, for every feasible unanswered question i_k and prior sequence $i_1, \ldots, i_{k-1}$, the question is chosen according to the rollout rule (2.4). The function H uses the greedy heuristic as the base policy, and its performance is approximated by the performance of an equivalent non-blocking quiz problem as described above.

5. The one-step rollout policy based on the index heuristic and certainty equivalence policy evaluation, where the function H in (2.4) uses the index heuristic as the base policy, and is approximated using the certainty equivalence approach described previously.

6. The selective two-step lookahead rollout policy based on the greedy heuristic, with certainty equivalence policy evaluation corresponding to an equivalent non-blocking quiz problem with horizon described as above.

7. The selective two-step lookahead rollout policy based on the index heuristic, with certainty equivalence policy evaluation corresponding to an equivalent non-blocking quiz problem with horizon described as above.

The problems selected for evaluation involve 20 possible questions and 20 stages, which are small enough so that exact solution using dynamic programming is possible. Associated with each question is a sequence of times, determined randomly for each experiment, when that question can be attempted. Floating point values were assigned randomly to each question from 1 to 10 in each problem instance. The probabilities of successfully answering each question were also chosen randomly, between a specified lower bound and 1.0. In order to evaluate the performance of the last six algorithms, each suboptimal algorithm was simulated 10,000 times, using independent event sequences determining which question attempts were blocked and which questions were answered correctly.

Our experiments focused on the effects of three factors on the relative performance of the different algorithms:

a) The lower bound on the probability of successfully answering a question, which varied from 0.2 to 0.8

b) The average percent of questions which can be answered at any one stage, which ranged from 10% to 50%.

c) The probability that individual question attempts will not be blocked, ranging from 0.3 to 1.0.

As in Section 4, for each experimental condition, we generated 30 independent problems and solved them with each of the 7 algorithms, and evaluate the corresponding performance using 10,000 Monte Carlo runs. The average performance is reported for each condition.

Minimum Probability of Success	0.2	0.4	0.6	0.8
Greedy Heuristic	54%	63%	73%	82%
Improvement by One-step Rollout	31%	26%	17%	6%
Improvement by Two-step Rollout	33%	26%	17%	6%
Index Heuristic	56%	67%	78%	84%
Improvement by One-step Rollout	30%	22%	12%	4%
Improvement by Two-step Rollout	31%	23%	12%	4%

Table 3: Performance of the different algorithms for stochastic quiz problems as the minimum probability of success of answering a question varies. The numbers reported are percentage of the performance of the optimal dynamic programming solution achieved, averaged across 30 independent problems.

The first set of experiments fixed the average percentage of questions which can be answered at a single stage to 10%, the probability that question attempts will not be blocked to 0.6, and varied the lower bound on the probability of successfully answering a question across four conditions: 0.2, 0.4, 0.6 and 0.8. Table 3 shows the results of our experiments. The average performance of the greedy and index heuristics in each condition are expressed in terms of the percentage of the optimal performance. The results for this experiment are very similar to the results we obtained earlier for deterministic quiz problems. Without rollouts, the performance of either heuristic is poor, whereas the use of one-step rollouts can recover a significant percentage of the optimal performance. As the risk associated with answering questions decreases, the performance of the heuristics improves, and the resulting improvement offered by the use of rollouts decreases. On average, the advantage of using selective two-step rollouts is small, but this advantage can be large for selected difficult problems.

Problem Density	0.1	0.3	0.5
Greedy Heuristic	54%	65%	78%
Improvement by One-step Rollout	31%	23%	13%
Improvement by Two-step Rollout	33%	24%	13%
Index Heuristic	56%	74%	87%
Improvement by One-step Rollout	30%	15%	5%
Improvement by Two-step Rollout	31%	16%	5%

Table 4: Performance of the different algorithms on stochastic quiz problems as the average number of questions per period increases. The numbers reported are percentage of the performance of the optimal dynamic programming solution achieved, averaged across 30 independent problems.

The second set of experiments fixed the lower bound on the probability of successfully answering a question to 0.2, and varied the average percent of questions which can be answered at any one stage across 3 levels: 10%, 30% and 50%. The results of these experiments are summarized in Table 4. As in the deterministic quiz problems, the performance of the greedy and index heuristics improves as the number of questions which can be answered at any one time approaches 100%. The results also show that, even in cases where the heuristics achieve good performance, rollout strategies offer significant performance gains.

The last set of experiments focused on varying the blocking probability that an attempt to answer a question at any one time is not blocked over 3 conditions: 0.3, 0.6 and 1.0. The last condition corresponds to the deterministic quiz problems of Section 3. Table 5 contains the results of these experiments. As the blocking probability increases, there is increased randomness as to whether questions may be available in the future. This increased randomness leads to improved performance of myopic strategies, as shown in Table 5. Again, the advantages of the rollout strategies are evident even in this favorable case.

Probability of Non-Blocking	0.3	0.6	1
Greedy Heuristic	73%	54%	41%
Improvement by One-step Rollout	17%	31%	34%
Improvement by Two-step Rollout	18%	33%	40%
Index Heuristic	75%	56%	43%
Improvement by One-step Rollout	16%	30%	34%
Improvement by Two-step Rollout	16%	31%	38%

Table 5: Performance of the different algorithms on stochastic quiz problems as the probability of non-blocking increases. The numbers reported are percentage of the performance of the optimal dynamic programming solution achieved, averaged across 30 independent problems.

The results in Tables 3, 4 and 5 provide ample evidence that rollout strategies offer superior performance for stochastic quiz problems, while maintaining polynomial solution complexity.

7. CONCLUSION

In this paper, we studied stochastic scheduling problems arising from variations of a classical search problem known as a quiz problem. We grouped these variations into two classes: the deterministic quiz problems, for which optimal strategies can be expressed as deterministic sequences, and the stochastic quiz problems, for which optimal strategies are feedback functions of the problem state. For either of these classes, the computational complexity of obtaining exact optimal solutions grows exponentially with the size of the scheduling problem, limiting the applicability of exact techniques such as stochastic dynamic programming.

In this paper, we develop near-optimal solution approaches for deterministic and stochastic quiz problems that are computationally tractable based on the use of rollout algorithms. For stochastic quiz problems, we introduced a novel approach to policy evaluation which resulted in polynomial complexity algorithms for obtaining near-optimal strategies. Our computational experiments show that these rollout algorithms can substantially improve the performance of index-based and greedy algorithms for both deterministic and stochastic quiz problems.

REFERENCES

[BBS95] Barto, A. G., Bradtke, S. J., and Singh, S. P., 1995. "Learning to Act Using Real-Time Dynamic Programming," Artificial Intelligence, Vol. 72, pp. 81-138.

[BTW97] Bertsekas, D. P., Tsitsiklis, J. N., and Wu, C., 1997. Rollout Algorithms for Combinatorial Optimization, Heuristics, (to appear).

[BeT96] Bertsekas, D. P., and Tsitsiklis, J. N., 1996. Neuro-Dynamic Programming, Athena Scientific, Belmont, MA.

[Ros83] Ross, S. M., 1983. Introduction to Stochastic Dynamic Programming, Academic Press, N. Y.

[TeG96] Tesauro, G., and Galperin, G. R., 1996. "On-Line Policy Improvement Using Monte Carlo Search," presented at the 1996 Neural Information Processing Systems Conference, Denver, CO.

[Whi82] Whittle, P., 1982. Optimization over Time, Vol. I, Wiley, N. Y.

[Whi83] Whittle, P., 1983. Optimization over Time, Vol. II, Wiley, N. Y.

APPENDIX B

Multiplatform Path Planning Under Uncertainty

Stephen D. Patek
David A. Castañon
David Logan

Abstract

We consider the problem of planning the paths of multiple vehicles in observing a battle space with the possibility of vehicle destruction. The model we develop is a special case of the more general scheduling problem described in [1]. Our goal is to illustrate the significant complexities which arise when stochastic effects (random vehicle destruction) are introduced into the model. To this end, we define a particular instance the scheduling problem which will serve as a testbed for various methodologies. Dynamic programming is the classical framework which characterizes solutions to our path planning problem and drives the algorithmic development. While computationally expensive, the dynamic programming recursion can be employed to solve the stochastic problem directly. Similarly, dynamic programming can be used to solve various deterministic auxiliary problems whose solutions provide heuristic solutions to the stochastic problem. Unfortunately, even though the auxiliary problems are less involved than the original stochastic problem, the computational requirements of brute-force dynamic programming are roughly equivalent. A more promising approach to solving the deterministic auxiliary problems is to relax the integer constraints and obtain quick solutions through proven linear programming and network optimization techniques. This paper will summarize the work done so far and will provide guidelines for future efforts.

1 Introduction

Planning and scheduling problems arise in many important Air Force operations, from tasking multi-platform missions for information collection to development of tactical air operations. In most of these problems, limited surveillance, logistics and combat resources are carefully scheduled to accomplish a set of missions. However, information regarding the battle space for the missions contains uncertainty, is incomplete, is not always correct, and evolves continuously. This has led to the development of a suite of sensors which are capable of providing real-time information on the progress of the missions, which can be used for replanning in response to new information. In order to exploit this information, new techniques are necessary for the real-time solution of large scheduling and planning problems in the presence of uncertainty.

In this paper, we consider a simple class of mission planning problems, involving the scheduling of multiple vehicles to perform multiple risky missions each. A general form of this model has been described in in [1]. In this model, vehicle motion is constrained to a directed graph, the nodes of which may contain missions which must be performed. Vehicles which visit a node may choose to perform the mission associated with that node, at the risk of being destroyed. The state of the problem at any stage consists of the location of the vehicles which are still alive and the time; future uncertainty enters into the state evolution as vehicles may be destroyed in risky situations.

From a mathematical perspective, this class of scheduling problems can be formulated as Markov decision problems [4]. These problems can be solved exactly using the stochastic dynamic programming algorithm. However, even for relatively small graphs, the amount of time required to find the optimal stochastic dynamic programming solution is far greater than allowed for real-time applications.

In this paper, we develop and evaluate several approximations to the exact stochastic dynamic programming algorithm which reduce the computation requirements while maintaining the feedback planning aspects of stochastic dynamic programming. The approximate algorithms are based on constructing approximations of the optimal cost-to-go function, in a manner similar to the methodology of neuro-dynamic programming [3]. In particular, we focus on rollout algorithms, a class of suboptimal solution methods inspired from the policy iteration methodology of DP and the approximate policy iteration methodology of neuro-dynamic programming. Algorithms of this type have been proposed by Tesauro [5] in the context of simulation-based computer backgammon.

Rollout algorithms were first proposed for the approximate solution of discrete optimization problems by Bertsekas and Tsitsiklis [3], and by Bertsekas, Tsitsiklis, and Wu [2], and the methodology developed here for the quiz problem strongly relates to the ideas in these sources. Generally, rollout algorithms are capable of magnifying the effectiveness of any given heuristic algorithm through sequential application. This is due to the policy improvement mechanism of the underlying policy iteration process.

In addition to rollout algorithms, we also discuss other approaches for obtaining approximate cost-to-go values. These approaches include using dynamic programming to solve various deterministic auxiliary problems whose solutions provide heuristic solutions to the stochastic problem, and relaxing integer constraints on the schedule to obtain quick solutions through linear programming and network optimization techniques.

We now outline the remainder of this report. In Section 2 we start with a verbal description of the path planning problem. This is followed by a more formal description in mathematical notation. In Section 3, we describe the solution approaches which we propose for the path planning problem. In Section 4, we present our experimental results. The main observation here is that the presence of uncertainty makes a difference: as would one expect, the qualitative nature of the optimal routing policy depends on the probabilities of vehicle destruction. We offer some brief conclusions in 6.

2 A Model for Multiplatform Path Planning

In this section we present a model which captures the inherent difficulty of multiplatform path planning with stochastic uncertainty. We consider the situation faced by a decisionmaker who requires more information about specific regions. We suppose that there is a finite set of regions of interest, each region corresponding to a node, along with a finite set of geographically significant way points, which are the positions (nodes) to which various reconnaissance platforms can be sent. We assume that the regions-of-interest and way points are connected by a finite number of arcs which are the relatively safe conduits through which reconnaissance vehicles may travel, forming a graph. The commander has access to a finite number of reconnaissance vehicles which can be routed to the various nodes on the graph in order to gain as much information as possible on a finite time-horizon. To keep the model simple, we assume that the reconnaissance vehicles are identical. In addition, we assume:

1. the vehicles respond to routing commands on a discrete stage basis,
2. they are able to transmit what they observe immediately upon encountering a region of interest,
3. they each require one unit of (discrete) time to traverse any arc on the graph,
4. for a given arc on the graph, each vehicle experiences the same arc-dependent risk of being destroyed (at a given cost), and
5. all vehicles are initially located at an origin node of the graph to which they must return by the end of the mission time (or else a penalty is paid).

When a region of interest is first visited the desired information about that region is gained deterministically, regardless of which vehicle arrives. (The value of the reconnaissance does not increase with the number of vehicles which arrive, if two or more arrive simultaneously.) We leave open the possibility that future revisits will also be worthwhile, depending on random events. The number of worthwhile visits will not be known a priori but will be revealed as a the mission progresses.

We now describe the model on a more formal level. Let $\mathcal{N} = \{0, 1, \ldots, N_N\}$ be the set of nodes of the graph (including regions of interest as well as way points). Let node 0 represent the origin node from which the vehicles depart and to which they should try to return (if possible). Generically, all nodes in $\mathcal{N}$ are regions of interest; however, some of these may always offer zero value for visits. For each node $i \in \mathcal{N}$, let $x_i(t) \in X_i$ represent the value-state of the node at time t (directly observed by the commander), which serves as an index determining the immediate value $v_i(x_i(t))$ of the *next* visit to node i. (Because the vehicle transmits its findings without delay, the reward is accrued immediately.) We assume that each node's value-state can take on only a finite set of values, and the value-state changes stochastically according to an acyclic Markov chain every time a vehicle arrives. Let P_i be the transition probability matrix for node i's value-state transition process.

Let $A(i) \subset \mathcal{N}$ be the subset of nodes j for which there is an arc from node i to j. This corresponds to the set of movement-options available to a vehicle located at node i. Let $B(i) \subset \mathcal{N}$ be the subset of nodes j for which there is an arc from j to node i. Note that we do not prohibit self-loops. That is, we may have $i \in A(i)$ (in which case it must also be true that $i \in B(i)$). Let N_A be the number of directed arcs in this graph representation of the problem. If the graph is symmetric, then $A(i) = B(i)$ for all $i \in \mathcal{N}$. Given i and $j \in A(i)$, p_{ij} represents the probability of surviving the transition from i to j. If the vehicle does not survive the transition, then it is lost at a cost of C_D. Otherwise, if a vehicle survives to the end of the mission, but only manages to make it back to node $i \in \mathcal{N}$, then a cost of $c_i \geq 0$ is incurred, where $c_0 = 0$. The goal is to control the movement of the vehicles on a time horizon of T stages, seeking to maximize the expected net value of the reconnaissance. We assume that N_V vehicles are available initially.

3 Solution Approaches

Our model for multiplatform path planning takes the form of a large scale stochastic optimal control problem to which the classical framework of dynamic programming applies. Even without the possibility of vehicle destruction, we have a very difficult optimization problem on our hands. Indeed, deterministic vehicle routing problems are recognized to be among the most difficult of optimization problems [6], and our introduction of stochastic effects such as random vehicle destruction, while well-motivated, only serves to complicate matters. Thus, since the methods of dynamic programming are feasible for only very small problems, we are faced with the necessity of using approximations.

A nice feature of our model is the fact that one can "build up" a solution for the case of n vehicles by first solving the (easier) problems involving $1, 2, \ldots, n-1$ vehicles, respectively. The reason for this is that the underlying controlled Markov chain is acyclic in the sense that once a vehicle is destroyed it cannot come back to life. This opens up the possibility of using *exact* solutions to endgames in a more general methodology involving approximate dynamic programming. Another interesting feature of the model is that there exist optimal policies which are "quasi-open loop" in the sense that the feedback is only important when vehicles are destroyed or when the value-states of regions of interest (randomly) change. Given a fixed number of remaining vehicles, the vehicles can be optimally routed by making a schedule of future transitions, and this plan would apply until a vehicle is destroyed or the value-state of a region of interest changes. Unfortunately, the formulation of such a plan will always involve accounting for the stochastic perturbations can take place in every time period, so the inherent difficulty of computing optimal solutions remains.

3.1 Stochastic Dynamic Programming

The model presented in Section 2 can be interpreted formally as a stochastic dynamic program. The reconnaissance "system" can be characterized by the number and locations of all of the vehicles and the value-states of all of the regions of interest. As a result, the number of possible system-states is finite, but large. In fact, the number of states is equal to ${N_N}^{N_V} \cdot \Pi_{i=1}^{N_N} |X_i|$, where $|X_i|$ is the number of value states for node i. Knowing the amount of time remaining in the mission and having access to the state of the system, the commander must choose from a finite number of routing commands for each of the remaining vehicles. Once a routing command has been issued, the system transitions to a new state subject to the random perturbations that are possible (eg. vehicle destruction and value-state transitions). The objective is to find the routing policy which maximizes the expected reward over the finite time horizon of the reconnaissance mission.

In theory, a backwards dynamic programming recursion may be used to compute the optimal cost-to-go function for this problem. The recursion first computes the optimal expected cost-to-go with one stage of the mission remaining and then uses this in a recursive fashion to compute the optimal expected cost-to-go with two stages remaining. Each step of the recursion proceeds similarly:

$$\begin{aligned} J_k^*[\xi(k)] &= \min_{u(k) \in U[\xi(k)]} \mathbf{E}\left\{ \quad g[\xi(k), \xi(k+1)] + J_{k+1}^*[\xi(k+1)] \quad \middle| \quad u(k), \xi(k) \quad \right\} && (1) \\ J_T^*[\xi(T)] &= h[\xi(T)] && (2) \end{aligned}$$

where

1. $\xi(k)$ is the state of the system at stage k,
2. $u(k)$ is a profile of routing command to the remaining vehicles at stage k,
3. $U[\xi(k)]$ is the set of combinations of routing commands available at state $\xi(k)$,
4. $g[\xi(k), \xi(k+1)]$ is the cost of vehicles destroyed in transitioning from $\xi(k)$ to $\xi(k+1)$,
5. $h[\xi(T)]$ is the cost associated with the locations of all surviving vehicles at the terminal state $\xi(T)$, and
6. the conditional expectation is over all possible outcomes $\xi(k+1)$ from $\xi(k)$.

The result of this recursion is a large lookup table which contains the optimal expected long term cost from every possible state of the system. Knowing the current state of the system $\xi(k)$ and having access to the optimal cost-to-go function allows us to pick optimal actions as the minimizers of the right hand side of Equation (1). An optimal policy $[\pi^* = \{\mu_0^*, \ldots, \mu_{T-1}^*\}]$ is obtained as a sequence of maps μ_k^* so that the action $\mu_k^*(\xi(k))$ minimizes the right hand side of Bellman's equation:

$$\mu_k^*[\xi(k)] \quad \in \arg \min_{u(k) \in U[\xi(k)]} \mathbf{E}\left\{ \quad g[\xi(k), \xi(k+1)] + J_{k+1}^*[\xi(k+1)] \quad \middle| \quad u(k), \xi(k) \quad \right\}. \tag{3}$$

While theoretically sound, the dynamic programming recursion is too expensive computationally to be used in real-world scheduling problems. This is due to the fact that large numbers of outcomes (transitions) are possible for each profile of routing commands and, even worse, the space of routing profiles grows combinatorially with the number of vehicles. Even a single step of the backwards recursion may be prohibitively expensive. Consequently, exact dynamic programming solutions are possible only for very small problems. Results of this type are presented in Sections 4.

3.2 Approximate Dynamic Programming via Deterministic Auxiliary Problems

Given the intractability of the multiplatform path planning problem, we are forced in practice to consider suboptimal routing policies for the vehicles. In this section we describe several potentially useful heuristics for path planning.

One type of heuristic to consider is one based on the solution to a deterministic auxiliary problem:

Auxiliary Problem 1 (Simple Deterministic Approximation) *Set the probabilities $p_{ij} = 1$ for all nodes i, j, and make the the value-state transition dynamics deterministic. Choose a schedule for the vehicles to minimize the (deterministic) cost over the finite planning horizon.*

Notice that the resulting problem, being deterministic, is considerably easier than the original stochastic problem of Section 2. Unfortunately, what remains is still a difficult optimization problem, resembling a vehicle routing problem where the vehicles are constrained to traveling along arcs of a graph. Unfortunately, dynamic programming seems to be the only framework for characterizing optimal solutions to this problem.

Ignoring the difficulty of obtaining solutions to Problem 1, a reasonable heuristic at a given state is to implement an action that would be optimal from the same state in the corresponding deterministic auxiliary problem.

Policy 1 *$[\pi^1 = \{\mu_0^1, \ldots, \mu_{T-1}^1\}]$ Compute the optimal policy $\bar{\pi}^* = \{\bar{\mu}_0^*, \ldots, \bar{\mu}_{T-1}^*\}$ for Auxiliary Problem 1. Set $\mu_k^1 = \bar{\mu}_k^*$ for all $k = 0, \ldots, T-1$.*

Another possibility is to think of the functions $\bar{J}_k^*$ as heuristic value-function approximations for the original stochastic model.

Policy 2 *$[\pi^2 = \{\mu_0^2, \ldots, \mu_{T-1}^2\}]$ Having access to the optimal cost-to-go functions $\bar{J}_k^*$ for Auxiliary Problem 1, choose each μ_k^2 such that $\mu_k^2[\xi(k)]$ is an action from state $\xi(k)$ at stage k which minimizes the right hand side of Bellman's equation, replacing the optimal cost to go function J_{k+1}^* with $\bar{J}_{k+1}^*$:*

$$\mu_k^2[\xi(k)] \quad \in \arg \min_{u(k) \in U[\xi(k)]} \mathbf{E}\left\{ \quad g[\xi(k), \xi(k+1)] + \bar{J}_{k+1}^*[\xi(k+1)] \quad \middle| \quad u(k), \xi(k) \quad \right\}. \tag{4}$$

(The minimum in this equation may not be unique.)

One way to see the difficulty of solving Auxiliary Problem 1 is to think of it in terms of network flows. It is possible to reformulate Auxiliary Problem 1 as an *integer* network flow problem with extra side constraints to restrict the amount of flow into nodes which offer value only for a limited number of visits. (We shall give such a formulation for the revised model of Section 5.) It turns out that by relaxing the integer constraints, the resulting optimal flows are fractional. Thus, solutions to Auxiliary Problem 1 cannot be obtained polynomially by means of solutions to linear programs.

3.3 Rollouts Based on Exact Evaluation of Heuristic Cost

Given an arbitrary (possibly suboptimal) policy $\pi = \{\mu_0, \ldots, \mu_{T-1}\}$, it is possible (in theory) to use the dynamic programming recursion to compute the expected cost-to-go for every state of the system. That is, by eliminating the min operation in Equation (1) and replacing $u(k)$ with $\mu_k[\xi(k)]$, we obtain the expected cost to go functions associated with π, denoted J_k^π, $k = 0, \ldots, T$. We obtain the effect of a policy iteration by "rolling it out" as follows.

Policy 3 (Exact Rollout) *[$\pi^3 = \{\mu_0^3, \ldots, \mu_{T-1}^3\}$] Choose each μ_k^3 such that $\mu_k^3[\xi(k)]$ is an action from state $\xi(k)$ at stage k which minimizes the right hand side of Bellman's equation, replacing the optimal cost to go function J_{k+1}^* with J_{k+1}^π:*

$$\mu_k^3[\xi(k)] \quad \in \quad \arg \min_{u(k) \in U[\xi(k)]} \mathbf{E}\left\{ \quad g[\xi(k), \xi(k+1)] + J_{k+1}^\pi[\xi(k+1)] \quad \middle| \quad u(k) \quad \right\}. \tag{5}$$

(The minimum in this equation may not be unique.)

This policy derives its name from the fact that, in practice, the evaluations $J_k^\pi[\xi(k), k]$ often are not computed precisely but are estimated through online Monte Carlo simulation of the original policy π. (Thus, the evaluations are made by "rolling-out" the die.) However, whenever we refer to Policy 3, we imply that the evaluations of the base policy are exact. Of course, exact evaluations are impractical for most real-world problems, so this technique only applies for very small problems. Results of this type are presented in Section 4.

4 Experimental Results, Part 1

We present in this section some computational results for the path planning problem. The experimentation we describe here is based on some initial implementations of the approaches described previously. In the following subsection we give a precise description of the path planning scenario used in these experiments. We follow that with a brief summary of our experimental findings.

The algorithms considered include the optimal dynamic programming algorithms for both the stochastic problem as well as Auxiliary Problem 1. It also computes (exactly) the expected cost-to-go function associated with Policy 1 and uses this to compute the (exact) cost-to-go associated with the rollout scheme of Policy 3.

4.1 An Instance of the Model

We consider an instance of the model in Section 2 involving two vehicles which must traverse the graph shown in Figure 1. The arcs in the graph are bidirectional, with thick lines representing risk-free transitions and thin lines representing transitions that are successful with the indicated probabilities. Nodes 5, 8, 9, 11, 12, and 13 are regions of interest, and the remaining nodes are simply geographic way points. Nodes 8 and 9 offer 50 points each for first visits, and 0 points for every other successive visit. Similarly, nodes 11 and 13 offer 75 points for first visits, and node 12 offers 150 points for a first visit. Node 5 is special in the sense that it offers 500 points for the first visit, and then with probability p offers 500 points for the second visit (and 0 points thereafter) and with probability $1-p$ offers 0 points for the second and all remaining visits. Thus, node 5 is the "big win" in this scenario, offering a minimum of 500 and up to 1000 points, to be determined immediately after the first visit. We impose a relatively short time horizon, between 8 and 20 stages, with a 200 point penalty for each vehicle that fails to return home, to force the decision maker to think carefully about how to proceed. Notice that the circuits $\{0,1,2,3,4,5,4,3,2,1,0\}$ and $\{0,6,7,10,11,12,13,10,7,6,0\}$ each involve 10 transitions. Thus, it is not clear a priori whether

1. to send the vehicles on separate tracks (a naive attempt to acquire most of the value),
2. to send vehicles along redundant tracks (to maximize the probability of making it to the most valuable node), or
3. to send vehicles out with some other routing philosophy in mind.

Notice that we allow self-loops only at nodes 0 and 5. In our experimental results we adjust the values of the parameters (p_1, p_2, p_3, p, and T) to see the qualitative nature of the various solutions. To be clear, in defining Auxiliary Problem 1 for this scenario, we set $p_1 = p_2 = p_3 = 0$ and $p = 1$.

4.2 Results

Figures 2 through 7 illustrate the relative performances and behavior of four different policies π^*, π^1, π^2, and π^3 for a number of different settings of the parameters p, p_1, p_2, p_3, and T. The policies are:

1. π^* is the optimal stochastic dynamic programming policy for this problem.
2. π^1 is the policy generated by the nave deterministic approximation.
3. π^2 is the policy obtained by approximate dynamic programming, using the deterministic cost of the nave approximation as a cost-to-go approximation.
4. π^3 is the policy obtained by rollout, using the exact cost-to-go evaluation associated with the nave deterministic approximation policy.

We consider value of T equal to 8, 12, 15, and 20, and for each value of T we consider

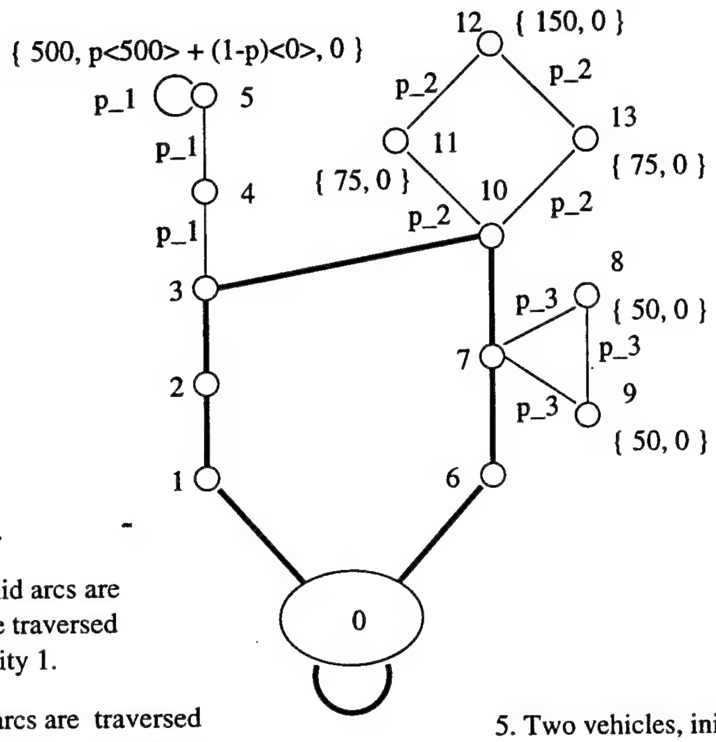
1. a "risky" case with $p_1 = .65, p_2 = .875$, and $p_3 = .975$ and
2. a relatively "risk-free" case with $p_1 = .9, p_2 = .95$, and $p_3 = .98$.

We keep the probability of a fruitful second visit to node 5 set to one half throughout (i.e. $p = .5$). The exact expected cost to go from the initial state (all vehicles at home at time zero with all regions unvisited) is displayed for each policy, along with the corresponding "ideal trajectory" which is the sequence of pairs of nodes visited by the vehicles given that neither of the vehicles are destroyed. These ideal trajectories indicate the *character* of the various policies under consideration. We notice in the data the following trends.

1. The naive policy π^1, which always implements an optimal action in the Auxiliary Problem 1, often does significantly worse than the optimal policy, especially in the risky cases.
2. The value-based policy π^2, which uses the optimal cost of Auxiliary Problem 1 as an approximation of the optimal cost-to-go function always does worse than π^1, often significantly worse. Perhaps the reason for this is that the approximation is state-insensitive. That is, if there is enough time left in the deterministic approximation, all of the value to be gained in the graph can be picked up with certainty.
3. The (exact) rollout policy π^3 recovers some (but not all of the) expected value achieved by π^*. The relative amount of recovery seems to diminish as the time horizon T increases.

We also notice that there is some insight to be gained in studying the ideal trajectories for each of the policies. For example, in Figure 2, the total cost of the ideal trajectory of π^* is -332, whereas the corresponding total cost for π^1 is -284. The explanation for this discrepancy lies in the relative riskiness of the π^1 routing. The optimal policy chooses to do redundant routing in order to account for the risk, whereas the nave policy branches the routing immediately, and thus requires three-extra medium risk transitions, compared to π^*. We also notice that the ideal trajectory of π^1 can be characterized by the "divide and conquer" philosophy, whereas π^* involves some redundancy (with vehicles following each other to ensure that some value is obtained). This philosophical dichotomy is even more pronounced in other "risky" cases.

A Scenario for New World Vistas



Notes:

1. All arcs are bidirectional.
2. The thick solid arcs are "safe" and are traversed with probability 1.
3. The dashed arcs are traversed with the indicated probabilities, where p_1 < p_2 < p_3.
4. The quantities enclosed by curly-brackets are the rewards associated with the first visits to the corresponding nodes. Note that after the first visit to the node initially worth 500 units the dice are rolled and with probability p a second visit will be worth 500 units again (and 0 units thereafter) and with probability (1-p) all revisits will be worth 0.
5. Two vehicles, initially.
6. 200 units/vehicle cost for destruction or not returning to node 0 within the time horizon.

Figure 1: A scenario for the path planning problem

Policy	CTG							
π^*	-332.2	(0, 0) (13, 3)	(0, 6) (10, 4)	(6, 7) (3, 5)	(7, 8) (2, 5)	(10, 9) (1, 4)	(11, 7) (0, 3)	(12, 10)
π^1	-284.4	(0, 0) (5, 10)	(0, 6) (4, 7)	(1, 7) (3, 8)	(2, 10) (2, 7)	(3, 11) (1, 6)	(4, 12) (0, 0)	(5, 13)
π^2	-206.1							
π^3	-304.1							

Figure 2: Results for the case that $p_1 = .65, p_2 = .875, p_3 = .975, p = .5$, and $T = 12$.

Policy	CTG							
π^*	-840.6	(0, 0) (4, 10)	(1, 6) (3, 7)	(2, 7) (2, 8)	(3, 10) (1, 7)	(4, 11) (0, 6)	(5, 12) (0, 0)	(5, 13)
π^1	-836.7	(0, 0) (5, 10)	(0, 6) (4, 7)	(1, 7) (3, 8)	(2, 10) (2, 7)	(3, 11) (1, 6)	(4, 12) (0, 0)	(5, 13)
π^2	-824.7							
π^3	-839.6							

Figure 3: Results for the case that $p_1 = .9, p_2 = .95, p_3 = .98, p = .5$, and $T = 12$.

Policy	CTG								
π^*	-378.3	(0, 0) (3, 5)	(6, 6) (2, 5)	(7, 7) (1, 4)	(10, 8) (0, 3)	(11, 9) (0, 2)	(12, 10) (0, 1)	(13, 3) (0, 0)	(10, 4) (0, 0)
π^1	-301.3	(0, 0) (4, 13)	(0, 0) (5, 10)	(0, 0) (5, 7)	(0, 6) (4, 8)	(0, 7) (3, 9)	(1, 10) (2, 7)	(2, 11) (1, 6)	(3, 12) (0, 0)
π^2	-201.4								
π^3	-315.3								

Figure 4: Results for the case that $p_1 = .65, p_2 = .875, p_3 = .975, p = .5$, and $T = 15$.

Policy	CTG								
π^*	-895.7	(0, 0) (3, 7)	(1, 6) (10, 8)	(2, 7) (7, 9)	(3, 10) (6, 7)	(4, 11) (0, 6)	(5, 12) (0, 0)	(5, 13) (0, 0)	(4, 10) (0, 0)
π^1	-867.2	(0, 0) (4, 13)	(0, 0) (5, 10)	(0, 0) (5, 7)	(0, 6) (4, 8)	(0, 7) (3, 9)	(1, 10) (2, 7)	(2, 11) (1, 6)	(3, 12) (0, 0)
π^2	-624.7								
π^3	-867.3								

Figure 5: Results for the case that $p_1 = .9, p_2 = .95, p_3 = .98, p = .5$, and $T = 15$.

Policy	CTG							
π^*	-391.5	(0, 0) (13, 3) (3, 0)	(0, 6) (10, 2) (2, 0)	(6, 7) (3, 1) (1, 0)	(7, 8) (4, 2) (0, 0)	(10, 9) (5, 3) (0, 0)	(11, 7) (5, 2) (0, 0)	(12, 10) (4, 1) (0, 0)
π^1	-301.266	(0, 0) (0, 0) (5, 10)	(0, 0) (0, 6) (5, 7)	(0, 0) (0, 7) (4, 8)	(0, 0) (1, 10) (3, 9)	(0, 0) (2, 11) (2, 7)	(0, 0) (3, 12) (1, 6)	(0, 0) (4, 13) (0, 0)
π^2	-201.4							
π^3	-315.3							

Figure 6: Results for the case that $p_1 = .65, p_2 = .875, p_3 = .975, p = .5$, and $T = 20$.

Policy	CTG							
π^*	-926.3	(0, 0) (5, 3) (1, 10)	(1, 6) (5, 10) (0, 3)	(2, 7) (4, 3) (0, 2)	(3, 8) (3, 10) (0, 1)	(2, 9) (2, 11) (0, 0)	(3, 7) (3, 12) (0, 0)	(4, 10) (2, 13) (0, 0)
π^1	-867.2	(0, 0) (0, 0) (5, 10)	(0, 0) (0, 6) (5, 7)	(0, 0) (0, 7) (4, 8)	(0, 0) (1, 10) (3, 9)	(0, 0) (2, 11) (2, 7)	(0, 0) (3, 12) (1, 6)	(0, 0) (4, 13) (0, 0)
π^2	-624.7							
π^3	-867.3							

Figure 7: Results for the case that $p_1 = .9, p_2 = .95, p_3 = .98, p = .5$, and $T = 20$.

5 A Restricted Model for Multiplatform Planning

In this section we discuss a special version of the model presented in Section 2. In this context, we restrict the value-state space X_i of each node i to having at most two states. If X_i has a single element, then every visit to i results in zero accumulated value. If a node i has two value states, then one state corresponds to the node being unvisited, offering value v_i for the next (first) visit, and the other state corresponds to the node having already been visited, offering zero value for all future visits. Note that the revised model has ${N_N}^{N_V} \cdot 2^{\bar{N}_N}$ states, where $\bar{N}_N$ is the number of nodes offering reconnaissance value (whose value-state spaces have two elements) .

Despite the simplification that the revised model affords, we are still hindered by the curse of dimensionality and need to consider methods that scale well with the size of the problem both in terms of graph complexity and numbers of vehicles. At this point we return to the notion of integer relaxations of network-flow approximations of the original problem. Since efficient LP solvers are available for such problems, fast heuristics can be derived based on such approximations which will then serve as base-policies in a rollout scheme. In the following, we consider one such network-flow approximation where we introduce side constraints to limit the amount of "value-flow" into regions of interest, and we introduce gains to model the "siphoning-off" of flow proportional to the probability of destruction along arcs. It is useful at this point to introduce some new notation.

1. Let $y^k_{m,ij}$ be the vehicle-m "value flow" that makes it from node i to node $j \in A(i)$ at stage k. (Such variables are defined for all $i \in \mathcal{N}$, $j \in A(i)$, $m = 1, \dots, N_V$, and $k = 0, \dots, T-1$.) This flow corresponds to first visits to nodes offering reconnaissance value.

2. Let $z^k_{m,ij}$ be the vehicle-m "valueless flow" that makes it from from i to node $j \in A(i)$ at stage k. (Such variables are defined for all $i \in \mathcal{N}$, $j \in A(i)$, $m = 1, \dots, N_V$, and $k = 0, \dots, T-1$.) This flow corresponds to visits to nodes offering no reconnaissance value.

3. Let $s_m(i)$ be an indicator function, taking on the value of one if vehicle m is initially located at node i.

Note that the quantity $(y^k_{m,ij} + z^k_{m,ij})/p_{ij}$ may be interpreted as the amount of vehicle-m flow *sent* from node i to j at stage k. Let y^k and z^k be vector representations of the flow variables at stage k, and let y and z represent aggregates of the flow variables for $k = 0, \dots, T-1$.

Auxiliary Problem 2 (Integer-relaxed Network Flow Reformulation)

$$\min_{(y,z)\in F} \sum_{m=1}^{N_V} \sum_{i\in\mathcal{N}} \left\{ C_D \left[\sum_{j\in A(i)} \sum_{k=0}^{T-1} \frac{(1-p_{ij})}{p_{ij}} (y^k_{m,ij} + z^k_{m,ij}) \right] + \sum_{j\in B(i)} \left[c_i (y^{T-1}_{m,ji} + z^{T-1}_{m,ji}) - v_i \sum_{k=0}^{T-1} y^k_{m,ji} \right] \right\}, \quad (6)$$

where F is the set of feasible flow vectors, characterized as follows.

1. *Nonnegativity:*
$$y^k_{m,ij} \geq 0 \quad and \quad z^k_{m,ij} \geq 0,$$
for all $i \in \mathcal{N}$, $j \in A(i)$, $m = 1, \dots, N_V$, and $k = 0, \dots, T-1$.

2. *Vehicle Flow Constraints:*
$$(y^k_{m,ij} + z^k_{m,ij})/p_{ij} \leq 1,$$
for all $i \in \mathcal{N}$, $j \in A(i)$, $m = 1, \dots, N_V$, and $k = 0, \dots, T-1$. (These constraints obviate the need to upper bound the $y^k_{m,ij}$ and $z^k_{m,ij}$.)

3. *Value Node Constraints:*
$$\sum_{k=0}^{T-1} \sum_{m=1}^{N_V} \sum_{j\in B(i)} y^k_{m,ji} \leq 1,$$
for all $i \in \mathcal{N}$. (These are the difficult side constraints.)

4. *Source Constraints:*
$$\sum_{j\in A(i)} (y^0_{m,ij} + z^0_{m,ij})/p_{ij} = s_m(i),$$
for all $i \in \mathcal{N}$ *and* $m = 1, \ldots, N_V$.

5. *Flow Conservation:*
$$\sum_{j\in B(i)} (y^k_{m,ji} + z^k_{m,ji}) = \sum_{j\in A(i)} (y^{k+1}_{m,ij} + z^{k+1}_{m,ij})/p_{ij},$$
for all $i \in \mathcal{N}$, $m = 1, \ldots, N_V$, *and* $k = 0, \ldots, T-2$.

This auxiliary problem amounts to a network flow problem with gains and side constraints, where we effectively split each node into two nodes (one which offers value but whose input arcs are capacity constrained and the other which offers no value but can be revisited often) and then make as many copies of the new graph as there are time periods remaining in the problem.

In the paragraph following Policy 4 we will discuss an important theoretical difficulty with this auxiliary problem which affects its ability to approximate the stochastic model of Section 5. First, however, there are some important (but more superficial) things to discuss about this formulation. To begin, the linear program we have defined involves $2N_V N_N N_A T$ variables and $N_V T(N_A + N_N) + N_N$ constraints. There are two features which prevent this from being an "easy" network flow problem: (1) the N_N side constraints (which couple the flows of the N_V vehicles) and (2) the gains (which model the probability of destruction). Individually, these factors can prevent an otherwise "normal" network flow problem from having fractional solutions. (Check.) If the p_{ij} are set to one, then the problem is a (difficult) network flow problem with side constraints, generically having fractional solutions. To solve Auxiliary Problem 2, we are compelled to use a generic LP solver, in this case CPLEX4.0. One nice feature of CPLEX is that it allows the user to input an initial basis for the primal simplex algorithm it implements. Specifically, since we will be solving many slightly different problems in the context of rollouts, we will be able to use the optimal solution of one problem as the initial basis for the next problem. (These "hot starts" have the potential to really speed up our heuristics.) Finally, we mention that to obtain an approximation of the optimal cost to go from any state of the system at any time t, it is only necessary to

1. replace T with $T - t$ to model the shorter time horizon,
2. to set to zero the right hand side of the side constraints corresponding to nodes that have already been visited, and
3. replace $s_m(\cdot)$ with the zero-function to model the fact that certain vehicles have been destroyed.

Auxiliary Problem 2 immediately gives rise to another heuristic policy.

Policy 4 *[$\pi^4 = \{\mu^4_0, \ldots, \mu^4_{T-1}\}$] Choose each μ^4_k such that $\mu^4_k[\xi(k)]$ is an action from state $\xi(k)$ at stage k which minimizes the right hand side of Bellman's equation, replacing the optimal cost to go function J^*_{k+1} with $val[LP(\xi(k+1), k+1)]$:*
$$\mu^4_k[\xi(k)] \quad \in \quad \arg \min_{u(k)\in U[\xi(k)]} \mathbf{E}\{ \quad g[\xi(k), \xi(k+1)] + val[LP(\xi(k+1), k+1)] \quad | \quad u(k), \xi(k) \quad \}. \qquad (7)$$
(The minimum in this equation may not be unique.)

Note that Policy 4 is analogous to Policy 2 in the sense that $val[LP(\xi(k), k)]$ is used as a heuristic value function approximation.

We now return to the theoretical difficulty about Auxiliary Problem 2 which we alluded to earlier. That is, with the $p_{ij} < 1$, the side constraints do not completely capture the restriction that one and only one vehicle can pick up value at a region of interest. In particular, it is possible for the flow of one vehicle to return to a region of interest to gain value over more than one time period. Suppose, for example, that $y^k_{m,ji} \in (0, 1)$. There is nothing which prevents the optimal flow vector from recycling some of that flow so that $y^{k+2}_{m,ji} > 0$. This is a possibility which is forbidden in the stochastic integer constrained version of the problem. (Note that this problem goes away if we set the p_{ij}'s to one in our definition of Auxiliary

Problem 2.) Because we have relaxed the integer constraints, not only will we get fractional flows, but these flows will generically correspond to vehicles visiting nodes for value more than once. Despite this difficulty, the auxiliary problem is of interest because (1) for many scenarios the approximation may not be that bad and (2) the fractional solutions of this problem can be used as a starting point in constructing feasible primal solutions that provide (hopefully) tight upper bounds to the optimal value of the stochastic problem. In pursuing the latter idea, there are a number of ways to proceed:

1. One possibility is to decompose the fractional solutions of Auxiliary Problem 2 into an enumerated list of feasible (integer) multipaths (paths for all of the vehicles) and then evaluate and sum the expected costs associated with each trajectory. The cost of the best such multipath should yield an accurate approximation of the cost of the optimal solution to the original stochastic problem.

2. Another possibility is to fix the fractional solution to Auxiliary Problem 2 by

   (a) choosing a vehicle who's flow should be repaired,

   (b) removing the flow corresponding to that vehicle and then resolving it's relaxed flow problem incorporating the extra constraint that this flow cannot visit value-nodes more than once (a heuristic may have to be employed here), and

   (c) checking the resulting multipath and determining whether to return to the first step or to quit.

   The point of this procedure is that it may be considerably easier (and faster) to enforce the the non-cycling constraint in solving single-vehicle problems than to try to fix the original fractional multipath by considering all vehicles at once. (The procedure above could also be used in starting from scratch, without having the solution to Auxiliary Problem 2 in hand.)

Of course, once we have identified a procedure to fix the solutions to Auxiliary Problem 2, we may use the costs of the new multipaths as approximations to the optimal cost-to-go in a value-based heuristic in a fashion similar to Policy 4.

Unfortunately, we have had almost no time to experiment with the above approaches. One observation we have made is that optimal solutions to Auxiliary Problem 2 tend to yield lower bounds which are way off compared to the optimal cost. (The lower bounds obtained are quite a bit lower than optimal.) This problem is even worse when we set the p_{ij}'s to one in Auxiliary Problem 2 (to obtain solutions to the deterministic approximation of the original problem). In fact, by implementing π^4 with the approximation where we set the p_{ij}'s to one we obtain ideal trajectories which simply don't make sense. This must correspond to the relatively poor performance of π^2 observed earlier. Apparently the optimal cost of the deterministic problem is a terrible approximation to the optimal cost-to-go of the original stochastic problem. In general, solutions to Auxiliary Problem 2 are bound to be overly optimistic simply because fractional solutions are allowed (which in turn gives rise to the flow-cycling problem we discussed earlier).

6 Conclusions

The multiplatform path planning problems described in this paper are very difficult stochastic combinatorial problems, despite the special structure which is built into the model. The cardinality of both the state space and the control space grow very rapidly with the numbers of vehicles and regions of interest. In the experimental results of Section 4, we saw that the introduction of random vehicle destruction has a big effect on the qualitative nature of optimal routing solutions. Our attempts to implement a reasonable and fast heuristic that will scale well with the size of the problem (cf. Section 5) are currently best characterized as "work in progress."

In terms of future directions, the ideas described in the paragraphs which follow Policy 4 need to be implemented and evaluated. In addition, one needs to develop approximate approaches for policy evaluation in large scenarios in order to use rollout algorithms. The computational cost of exact policy evaluation grows exponentially with problem size, and quickly exceeds the available processing time. These approximate techniques can be based on Monte Carlo methods or other similar sampling approaches, and are currently under investigation.

References

[1] D. A. Castanon, D. A. Logan, and D. P. Bertsekas, "New World Vistas Proposal AFOSR BAA 97-1 Planning and Scheduling: Dynamic Assignment and Scheduling with Contingencies." Alphatech, Inc. Burlington, MA 01803.

[2] D. P. Bertsekas and J. N. Tsitsiklis and C. Wu "Rollout Algorithms for Combinatorial Optimization," *Heuristics,* (to appear).

[3] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming,* Athena Scientific, Belmont, MA, 1996.

[4] D. P. Bertsekas, *Dynamic Programming and Optimal Control,* Vols. I-II, Athena Scientific, Belmont, MA 1995.

[5] G. Tesauro and G. R. Galperin, "On-Line Policy Improvement Using Monte Carlo Search," presented at the *1996 Neural Information Processing Systems Conference,* Denver, CO.

[6] M. M. Solomon, "Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints," *Operations Research,* vol.35, no.2, March-April (1987), 254-265.